Name

intro – introduction to the Kerberos subroutines

Syntax

#include <krb.h>

#include <des.h>

cc [*options* **]** *files* **-lkrb -lknet**
-ldes -lacl [*libraries* **]**

Description

The Kerberos subroutines can provide for the authentication of and protection against the unauthorized modification of every message sent accross a TCP/IP network from one application to another. In addition, they provide a means to provide for the creation of access control lists (ACL) which an application can use with Kerberos authentication, to determine if another application is authorized to perform a particular action.

The `krb_svc_int(3krb)` routines are designed to initialize the Kerberos libraries so that the other Kerberos routines can function properly. The `krb_svc_init` routines are used to contact a Kerberos server to obtain a ticket-granting ticket that can be used by the `kerberos(3krb)`, `krb_sendmutual(3krb)`, and `krb_sendauth(3krb)` routines. They also initialize pieces of Kerberos library data. To use these routines, the libraries `libkrb.a`, `libknet.a`, and `libdes.a` must be linked with your application in the order listed.

The `kerberos(3krb)` routines `krb_mk_req` and `krb_rd_req` are designed to provide for the initial authentication of an application to another. They are designed to be used with applications that support "on-the-wire" protocols in which authentication information can be placed. The `kerberos(3krb)` routines `krb_mk_safe` and `krb_rd_safe` are designed to provide for the authentication of and protection against the modification of every message sent between two applications after the initial authentication message. To use these routines, the libraries `libkrb.a`, `libknet.a`, and `libdes.a` must be linked with your application in the order listed.

The `krb_sendmutual(3krb)` routines are designed to provide for the mutual authentication of two applications after the initial authentication of one application, X to another, Y. To provide mutual authentication, Y's identity is proven by the `krb_sendmutual` routines to X. To use these routines, the libraries `libkrb.a`, `libknet.a`, and `libdes.a` must be linked with your application in the order listed.

The `krb_sendauth(3krb)` routines are designed to provide both the initial authentication that `krb_mk_req` and `krb_rd_req` provide, as well as the mutual authentication of the `krb_sendmutal` routines. The `krb_sendauth` routines are designed to be used with applications that do not have room in the protocols they support for authentication information. To use these routines, the libraries `libkrb.a`, `libknet.a`, and `libdes.a` must be linked with your application in the order listed.

The krb_get_lrealm(3krb) routines are designed to provide information to the user about the Kerberos environment. To use these routines, the library libkrb.a must be linked with your application.

The des_crypt(3krb) routines are designed to provide support for the above routines with respect to Data Encryption Standard (DES) keys. The des_quad_cksum routine can be used to provide support for the authentication of and protection against the modification of every message sent between two applications after the initial authentication message. It is designed to be used only with applications that have room in their "on-the-wire" protocol for authentication information. To use these routines, the library libdes.a must be linked with your application.

The krb_set_tkt_string(3krb) routines are designed allow the user of the Kerberos libraries to modify some of the default settings of the Kerberos libraries. To use these routines, the library libkrb.a must be linked with your application.

The acl_check(3krb) routines are designed to provide for the creation and use of access control lists (ACL). After an application, X, correctly authenticates the identity of another, Y, the application X has the ability to assign access rights to Y, based on Y's identity. The routines above provide for the authentication of applications while the acl_check(3krb) routines provide the ability to store the access rights associated with each application. To use these routines, the library libacl.a must be linked with your application.

Files

/usr/lib/libkrb.a

/usr/lib/libknet.a

/usr/lib/libdes.a

/usr/lib/libacl.a

See Also

All the other Kerberos reference pages:

acl_check(3krb)
des_crypt(3krb)
kerberos(3krb)
krb_get_lrealm(3krb)
krb_sendauth(3krb)
krb_sendmutual(3krb)
krb_set_tkt_string(3krb)
krb_svc_init(3krb)
krb.conf(5krb)
krb_slaves(5krb)
krb_dbase(5krb)
ext_srvtab(8krb)
kdb_destroy(8krb)
kdb_edit(8krb)
kdb_init(8krb)
kdb_util(8krb)
kdestroy(8krb)

kerberos(8krb)
kinit(8krb)
klist(8krb)
kprop(8krb)
kpropd(8krb)
kstash(8krb)

Name

acl_check – Access control list (ACL) library routines.

Syntax

cc <files> -lacl -l krb

#include <krb.h>

acl_canonicalize_principal (*principal, buf*)
char **principal*;
char **buf*;

acl_check (*acl_file, principal*)
char **acl_file*;
char **principal*;

acl_exact_match (*acl_file, principal*)
char **acl_file*;
char **principal*;

acl_add (*acl_file, principal*)
char **acl_file*;
char **principal*;

acl_delete (*acl_file, principal*)
char **acl_file*;
char **principal*;

acl_initialize (*acl_file, mode*)
char **acl_file*;
int *mode*;

kname_parse (*primary_name, instance_name, realm_name, principal*)
char **primary_name*;
char **instance_name*;
char **realm_name*;
char **principal*;

Arguments

principal The name of a principal. Principal names consist of from one to three fields. The first field must be included because it stores the primary name of the principal. The second field is not always required. It begins with a period (.), and stores the instance name of the principal. The third field is not always requred. It begins with an "at" sign (@), and stores the realm name of the principal. The principal name format can be expressed as:

```
name[.instance][@realm]
```

For example, all of the names below are legitimate principal names:

```
venus
venus.root
venus@dec.com
venus.@dec.com
venus.root@dec.com
```

buf Pointer to the buffer that stores the canonical form of a principal name. The canonical form is derived from the form of a principal name. Like a principal name, it includes a primary name in its first field. Unlike a principal name, it must include an instance name as its next field even if the instance name is blank. Also, unlike a principal name, it must contain a realm field. If a canonical name is derived from a principal name that has no realm field, the local realm returned by `krb_get_lrealm(3krb)` is used as the realm field in the canonical name. Of the above examples, only the last two are in canonical form.

acl_file The path name of the file in which the access control list (ACL) is stored.

mode If the ACL file, *acl_file*, does not currently exist when `acl_initialize` is called, the file *acl_file*, is created with read, write, and access mode bits set equal to *mode*.

primary_name
The primary name portion of *principal*, returned by `kname_parse`. ANAME_SZ bytes of storage space must be allocated for *primary_name*.

instance_name
The instance name of *principal*, returned by `kname_parse`. INST_SZ bytes of storage space must be allocated for *instance_name*.

realm_name
The realm name of *principal*, returned by `kname_parse`. REALM_SZ bytes of storage space must be allocated for *realm_name*.

Description

The routines of the `acl_check` library allow you to perform various administrative functions on an access control list (ACL). An ACL is a list of Kerberos principals in which each principal is represented by a text string. The routines of this library allow application programs to refer to named ACLs to test whether a principal is a member of an ACL, and to add or delete principals from the ACL file.

The routines of the `acl_check` library are:

acl_canonicalize_principal
Stores the canonical form of the principal name pointed to by *principal* in the buffer pointed to by *buf*. This buffer must contain enough space to store a full canonical principal name (MAX_PRINCIPAL_SIZE characters). No meaningful value is returned by `acl_canonicalize_principal`.

acl_check
Verifies that the principal name, *principal*, appears in the ACL file, *acl_file*. This routine returns a zero (0) if the principal does not appear in the ACL, or if there is an error condition. If the principal is a member of the ACL, a one (1) is returned. The `acl_check` routine always canonicalizes a principal before trying to find it in the ACL. `acl_check` will determine if there is an ACL entry in the *acl_file* which exactly matches principal, *principal*, or if *principal* matches an ACL entry which contains a wildcard. A wildcard appears in place of a field name in an ACL entry and is represented as an asterisk (*). A wildcard in a field name of an ACL entry allows the ACL entry to match a principal name

that contains anything in that particular field. For example, if there is an entry, `venus.*@dec.com` in the ACL, the principals, `venus.root@dec.com`, `venus.@dec.com`, and `venus.planet@dec.com` would be included in the ACL. The use of wildcards is limited, for they may be used in only the three following configurations in an ACL file:

```
name.*@realm
*.*@realm
*.*@*
```

acl_exact_match
Verifies that principal name, *principal*, appears in the ACL file, `acl_file`. This routine returns a zero (0) if the principal does not appear in the ACL, or if any error occurs. If the principal is a member of the ACL, `acl_exact_match` returns a non-zero. The `acl_exact_match` routine does not canonicalize a principal before the ACL checks are made, and it does not support wildcards. Only an exact match is acceptable. So, for example, if there is an entry, `venus.*@dec.com` in the ACL, only the principal `venus.*@dec.com` would match the ACL entry. This routine makes it easy to find ACL entries with wildcards.

acl_add Adds the principal name, *principal*, to the ACL file, *acl_file*. This routine returns a zero (0) if it successfully adds the principal to the ACL. Otherwise, if there was an internal error, or if the principal is already in the ACL, the `acl_add` routine returns a non-zero value. The `acl_add` routine canonicalizes a principal, but treats wildcards literally.

acl_delete
Deletes the principal, *principal*, from the ACL file, *acl_file*. The routine returns a zero (0) if it successfully deletes the principal from the ACL. Otherwise, if there was an internal error or if the principal is not in the ACL, the `acl_delete` routine returns a non-zero value. The `acl_delete` routine canonicalizes a principal, but treats wildcards literally.

acl_initialize
Initializes the ACL file, *acl_file*. If the named *acl_file* does not exist, `acl_initialize` creates one with the permissions specified by the *mode* argument. If the ACL exists, `acl_initialize` removes all previously stored principal members of the list. This routine returns a zero (0) if successful or a nonzero if it fails.

kname_parse
parses the principal name, *principal*, and stores the primary name of the principal in *principal_name*, the instance name of the principal in *instance_name*, and the realm name of the principal in *realm_name*. `kname_parse` returns KNAME_FMT if the principal name is incorrectly formatted or if it is too long to be a principal name. It returns KSUCCESS if the parsing of the principal name succeeded.

See Also

kerberos(3krb), krb_get_lrealm(3krb)

des_crypt(3krb)

Name

des_crypt – Data Encryption Standard (DES) encryption library routines.

Syntax

```
#include <des.h>

int des_string_to_key    (str, key)
char          *str;
C_Block       *key;

int des_is_weak_key (key)
C_Block       key;

unsigned long des_quad_cksum  (input, output, length,
                               iterations, seed)
unsigned char *input;
unsigned long *output;
long          length;
int           iterations;
C_Block       *seed;

int des_key_sched  (key, schedule)
C_Block       key;
Key_schedule schedule;
```

Arguments

key	For `des_string_to_key`, *key* is a pointer to a `C_Block` of 8-byte length. For `des_quad_cksum`, `des_is_weak_key`, and `des_key_sched`, *key* is a pointer to a DES key.
str	A string that is converted to an 8-byte DES key.
input	Pointer to a block of data to which a quadratic checksum algorithm is applied.
output	Pointer to a pre-allocated buffer that will contain the complete output from the quadratic checksum algorithm. For each iteration of the quadratic checksum applied to the input, eight bytes (two longwords) of data are generated.
length	Length of the data to which the quadratic checksum algorithm will be applied. If input contains more than *length* bytes of data, then the quadratic checksum will only be applied to *length* bytes of input.
iterations	The number of iterations of the `des_quad_cksum` algorithm to apply to *input*. If output is NULL, then one iteration of the algorithm will be applied to *input*, no matter what the value of *iterations* is. The maximum number of iterations is four.
seed	An 8-byte quantity used as a seed to the *input* of the `des_quad_cksum` algorithm.
schedule	A representation of a DES key in a form more easily used with encryption algorithms. It is used as input to the `krb_sendmutual` routines.

Description

The des_crypt routines are designed to provide the cryptographic routines which are used to support authentication. Specifically, des_quad_cksum and des_key_sched are designed to be used with the DES key which is shared between one Kerberos principal and its authenticated peer to provide an easy authentication method after the initial Kerberos authentication pass. des_string_to_key and des_is_weak_key are designed to enable the input and inspection of a key by a user before that key is used with the Kerberos authentication routines. The des_crypt routines are not designed for general encryption.

The library makes extensive use of the locally defined data types C_Block and Key_schedule. The C_Block struct is an 8-byte block used by the various routines of the des_crypt library as the fundamental unit for DES data and keys.

Routines

string_to_key

Converts a null-terminated string of arbitrary length to an 8-byte, odd-byte-parity DES key. The *str* argument is a pointer to the character string to be converted and *key* points to a C_Block supplied by the caller to receive the generated key. The one-way function used to convert the string to a key makes it very difficult for anyone to reconstruct the string from the key. No meaningful value is returned.

des_is_weak_key

des_is_weak_key checks a new key input by a user to determine if it belongs to the well known set of DES keys which do not provide good cryptographic behavior. If a key passes the inspection of des_is_weak_key, then it can be used with the des_quad_cksum routine. The input is a DES key and the output is equal to 1 if the key is not a safe key to use; it is equal to 0 if it is safe to use.

des_quad_cksum

Produces a checksum by chaining quadratic operations on cleartext data. des_quad_cksum can be used to produce a normal quadratic checksum and, if used with the DES key shared between two authenticated Kerberos principals, it can also provide for the integrity and authentication protection of data sent from one principal to another.

Input of *length* bytes are run through the des_quad_cksum routine *iterations* times to produce *output*. If *output* is NULL, one iteration is performed and *output* is not affected. If *output* is not NULL, the quadratic checksum algorithm will be performed *iterations* times on input, placing eight bytes (two longwords) of result in *output* for each iteration. At all times, the low-order bits of the last quadratic checksum algorithm pass are returned by des_quad_cksum.

The quadratic checksum algorithm performs a checksum on a few bytes of data and feeds the result into the algorithm as an addition input to the checksum on the next few bytes. The seed serves as the additional input for the first checksum operation and, therefore, the final checksum that results depends upon the seed input into the algorithm. If the DES key shared between two Kerberos principals is used as the initial seed, then

since the checksum that results depends upon the seed, the ability to produce the checksum proves identity and authentication. Also, since the message cannot be altered without knowledge of the seed, it also provides for data integrity.

des_key_sched

`des_key_sched` is used to convert the key input into a new format that can be used readily with encryption functions. The result, schedule, can be used with the `krb_sendmutual` functions to enable mutual authentication of two Kerberos principals.

0 is returned from `des_key_sched` if sucessful.

-1 is returned if the each byte of the key does not have odd parity.

-2 is returned if the key is a weak key as defined by `des_is_weak_key`.

Name

kerberos – Kerberos authentication library routines

Syntax

```
#include <des.h>
#include <krb.h>

int krb_mk_req(tkt_authen_out, f_service, f_instance,
                f_realm, checksum)
KTEXT       tkt_authen_out;
char        *f_service;
char        *f_instance;
char        *f_realm;
u_long      checksum;

int krb_rd_req(tkt_authen_in, l_service, l_instance,
                f_hostaddr, ad, srvtab_file)
KTEXT       tkt_authen_in;
char        *l_service;
char        *l_instance;
u_long      f_hostaddr;
AUTH_DAT    *ad;
char        *srvtab_file;

int krb_get_cred(f_service, f_instance,
                f_realm, cred)
char        *f_service;
char        *f_instance;
char        *f_realm;
CREDENTIALS     *cred;

long   krb_mk_safe(in, out, in_length, key,
                l_addr, f_addr)
u_char              *in;
u_char              *out;
u_long              in_length;
C_Block             *key;
struct sockaddr_in  *l_addr;
struct sockaddr_in  *f_addr;

long   krb_rd_safe(in, in_length, key, f_addr,
                l_addr, msg_data)
u_char              *in;
u_long              in_length;
C_Block             *key;
struct sockaddr_in  *f_addr;
struct sockaddr_in  *l_addr;
MSG_DAT             *msg_data;
```

Arguments

f_service Character pointer to the primary name of the foreign principal. The local principal is the principal that calls the routines listed above. The local principal tries to communicate with the foreign principal.

f_instance Character pointer to the instance name of the foreign principal.

f_realm Character pointer to the realm name of the foreign principal.

l_service Character pointer to the primary name of the local principal.

l_instance Character pointer to the instance name of the local principal.

tkt_authen_out
Pointer to the text structure in which the Kerberos library routines build the ticket-authenticator pair. This structure is designed to be sent to the foreign principal to authenticate the local principal's identity to the foreign principal. Storage must be allocated for *tkt_authen_out*.

tkt_authen_in
Pointer to the ticket-authenticator pair that the Kerberos library uses to authenticate the foreign principal to the local principal. The data in this structure must have been generated by a call to `krb_mk_req` by the foreign principal and transmitted by the foreign principal to the local principal.

checksum The *checksum* parameter is input to `krb_mk_req`. It is packaged with the ticket-authenticator pair that is sent to the foreign principal. The *checksum* serves as a secret piece of data that can be known only to the foreign principal if the foreign principal is authenticated as the foreign principal. It is used to facilitate mutual authentication with `krb_sendmutual` and `krb_recvmutual`. See `krb_sendmutual(3krb)` for information about these two routines.

f_hostaddr Address of the machine from which the foreign principal sent the *tkt_authen_in* data.

f_addr Address of the socket that the foreign principal is using to communicate with the local principal.

l_addr Address of the socket that the local principal is using to communicate with the foreign principal.

ad Pointer to the AUTH_DAT structure that describes the authentication association between the local and foreign principals. The *ad* structure is output from `krb_rd_req`. You must allocate space for the *ad* structure.

srvtab_file The path name of the file that contains the key of the principal obtaining a ticket. If this value is set equal to a string of zero length, `srvtab_file[0]='\0'`, the default service table (srvtab) file is used. If this value is set equal to the NULL

pointer, then the key of the service is not read from the srvtab file, but is read from storage space internal to the libraries. The *srvtab_file* parameter cannot be set equal to the NULL string on the first call to `krb_rd_req`. The default srvtab file value is set to `/etc/srvtab`, although this value can be changed by a call to the `krb_set_srvtab_string` function. (See the `krb_set_tkt_string(3krb)` reference page).

key Pointer to the C_Block input to `krb_mk_safe` and `krb_rd_safe`. It contains a Data Encryption Standard (DES) key. The key that is usually used is the session key between the local and foreign principal.

cred A pointer to a credentials structure that is allocated by the caller of `krb_get_cred` and filled with data by `krb_get_cred`. The credentials structure includes the ticket that the local principal uses to authenticate the foreign principal. It also includes other authentication information associated with the foreign principal.

in Character pointer to the user data that must be included in a safe message.

out Character pointer to the safe message output by `krb_mk_safe`. The *in* parameter may not overlap with *out*.

in_length Length of the user data, *in*.

msg_data The *msg_data* parameter is a pointer to a MSG_DAT structure which must be allocated by the caller of `krb_rd_safe` and which is filled by `krb_rd_safe` with information about the safe message. A pointer to the user data sent within the safe message is also included in *msg_data*.

Description

The `krb_mk_req` calls are designed to be used by two principals that are attempting to authenticate themselves for the first time as well as by two principals that have authenticated once, but wish to authenticate all data passed between them.

The `krb_mk_req` and `krb_rd_req` routines are designed to be used by applications that communicate over a network, require the authentication of both parties across the communication path, and support "on-the-wire" protocols in which authentication data can be placed. These routines perform only the authentication of the first message sent between such applications. `krb_mk_req` creates a ticket-authenticator pair that can be included in the "on-the-wire" protocol of an application, and `krb_rd_req` reads the ticket-authenticator pair.

The `krb_mk_safe` and `krb_rd_safe` routines are used by applications that require that every message passed between them be authenticated and free from unauthorized modifications, and whose "on-the-wire" protocol has no room for authentication data. These routines only provide for the authentication and integrity protection of a message if the first authenticated message has already been sent by the `krb_mk_req/krb_rd_req` pair or the `krb_sendauth/krb_recvauth` pair. See `krb_sendauth(3krb)` for more information about the latter pair.

The `krb_mk_safe` routine encapsulates user data inside the `krb_mk_safe` "on-the-wire" message authentication protocol. `krb_rd_safe` can interpret the message authentication protocol and the message, and return the data encapsulated by `krb_mk_safe`. Since any application which is modified to use `krb_mk_safe` or

krb_rd_safe must encapsulate its "on-the-wire" protocol within the "on-the-wire" protocol of krb_mk_safe, the application must develop a method of distinguishing between the old and new "on-the-wire" protocols.

The des_quad_cksum routine (see des_crypt(3krb)) can be used to provide some of the guarantees of the krb_mk_safe and krb_rd_safe routines without encapsulating the protocol of the application.

The routines of this library make extensive use of the following locally defined data types: KTEXT, AUTH_DAT, CREDENTIALS, C_Block, and MSG_DAT. For specific information on the definitions of these data types, see the des.h and krb.h files.

Routines and Structures

krb_mk_req

Used to produce the data necessary to authenticate a principal "A" to a principal "B". It takes as input a checksum and the primary name, instance name, and realm name of the service to which the principal "A" is attempting to authenticate itself. krb_mk_req outputs a text structure in which the ticket to communicate with principal "B" and an authenticator have been combined to form a ticket-authenticator pair.

The application "A" must pass the ticket-authenticator pair to the principal "B" where it can be read by krb_rd_req. Once the ticket-authenticator pair has been read and verified, "A" has been authenticated to "B". Unless an attacker possesses the session key contained in the ticket, the attacker will be unable to modify or replay the ticket-authenticator pair.

The checksum can be used with krb_sendmutual and krb_recvmutual to provide for the authentication of "B" to "A" after krb_rd_req authenticates "A" to "B". Although the checksum value can be any value known only to "A", it is recommended that the checksum value used differ every time krb_mk_req is called. The following is a list of the return values from krb_mk_req and, if they are error codes, their possible cause:

KFAILURE	/etc/krb.conf file (see krb.conf(5krb)) cannot be opened, or it is not properly formed.
NO_TKT_FIL	The ticket file does not exist.
TKT_FIL_ACC	The ticket file cannot be opened or the ticket file cannot be accessed.
TKT_FIL_LCK	The ticket file could not be locked for access.
TKT_FIL_FMT	The ticket file format is incorrect.
AD_NOTGT	There is no ticket-granting ticket in the ticket file that can be used to ask for a ticket to communicate with the foreign principal.
SKDC_CANT	A Kerberos server must be contacted so that krb_mk_req can perform its function, but the attempt cannot be made because a socket cannot be opened or bound, or because there is no Kerberos server listed in /etc/krb.conf.
SKDC_RETRY	A Kerberos server needs to be contacted, but none responded even after several attempts.

INTK_PROT Kerberos protocol error.

KSUCCESS All went well.

krb_rd_req

This routine is used to read the authentication data produced by principal "A" with `krb_mk_req` and sent by "A" to principal "B". It takes as input the primary name and instance name of the local principal "B", as well as the authentication data sent to "B", the address of the machine from which "A" sent the ticket-authenticator pair, and the name of the file in which to find the key of the local principal. If the authentication attempt is successful, `krb_rd_req` will fill the *ad* structure with data about the authenticated association between "A" and "B".

The `krb_rd_req` routine returns zero (RD_AP_OK) upon successful authentication. If a packet was forged, modified, or replayed, then authentication fails.

The following is a list of the error values returned from `krb_mk_req` and their possible causes:

RD_AP_VERSION
: The versions of Kerberos used by the caller of `krb_mk_req` is incompatible with the `krb_rd_req` version.

RD_AP_MSG_TYPE
: The ticket-authenticator pair given to `krb_rd_req` was not actually a ticket-authenticator pair.

RD_AP_UNDEC
: The ticket was indecipherable. This error can be caused by a forged or a modified message.

RD_AP_INCON
: The message given to `krb_rd_req` contains an internal inconsistency. This could occur if the ticket in the ticket-authenticator pair does not match the authenticator.

RD_AP_BADD
: The ticket-authenticator pair cannot be used from the address, *f_hostaddr*.

RD_AP_TIME
: The authenticator in the ticket-authenticator pair is too old to be used to authenticate the foreign principal.

RD_AP_NYV
: The time at which the ticket of the ticket-authenticator pair was created, is too far ahead of the time of the local host of the local principal.

RD_AP_EXP
: The ticket is too old to be used.

krb_get_cred

Searches the caller's ticket file for the authentication information associated with the principal specified by the *f_service*, *f_instance*, and *f_realm*. If `krb_get_cred` finds information in the ticket file, it fills a credentials structure with the information and returns the status, GC_OK.

The following is a list of the error values returned from `krb_mk_req` and their possible causes:

NO_TKT_FIL The ticket file does not exist.

TKT_FIL_ACC	The ticket file cannot be opened or the ticket file cannot be accessed.
TKT_FIL_LCK	The ticket file could not be locked for access.
TKT_FIL_FMT	The ticket file format is incorrect.
GC_NOTKT	Information concerning the principal does not exist in the ticket file.

krb_mk_safe

Creates an authenticated but unencrypted message from text pointed to by *in*, of a length indicated by *in_length*. The routine uses the private session key (**key*) to seed the checksum algorithm, `des_quad_cksum`, that it uses as part of the authentication process. (For more information about `des_quad_cksum`, see the `des_crypt(3krb)` reference page.) The `krb_mk_safe` routine also uses the arguments *l_addr* and *f_addr* for authentication purposes.

A safe message does not provide privacy, but does provide protection against modifications in addition to providing authentication. The encapsulated message and header produced by `krb_mk_safe` are placed in the area pointed to by *out*. The routine returns the length of the output or a negative one (–1), indicating an error.

krb_rd_safe

Authenticates a received `krb_mk_safe` message and writes the appropriate fields in the message data structure **MSG_DAT**. The argument *in* points to the beginning of the received message. The argument *in_length* specifies the length of the message. The `krb_rd_safe` routine uses the private session key (**key*) to seed the `des_quad_cksum` routine (see the `des_crypt(3krb)` reference page) as part of its authentication process. The routine fills in the following **MSG_DAT** fields:

MSG_DAT Field	Description
`app_data`	Pointer to the application data
`app_length`	Length of the `app_data`
`time_sec`	Timestamp of the message in seconds
`time_5ms`	Timestamp of the message in 5-millisecond units
`swap`	A 1 if the byte order of the receiver is different from that of the sender

Note that the application must still determine if it is appropriate to byte-swap application data; the Kerberos protocol fields are already taken care of.

The `krb_rd_safe` routine returns RD_AP_OK if the message, *in*, is authenticated and has not been modified when it was sent between the foreign and the local principal. It is up to the caller to check the time sequence of messages and to check against recently replayed messages. The following is a list of the error values returned by `krb_rd_req` and their possible causes:

-1	A system call used by `krb_rd_safe` returned an error.
RD_AP_VERSION	The Kerberos version of the `krb_mk_safe` code that generated message, *in*, is not supported by the `krb_rd_safe` version used.

RD_AP_MSG_TYPE
: The message, *in*, is not really a message produced by `krb_mk_safe`.

RD_AP_MODIFIED
: The address of the machine from which *in* was sent does not match the address of the machine on which the `krb_mk_safe` message, *in*, was generated, or
The message was modified when it was sent from the foreign to the local principal, or
The message, *in*, is too small to be the message produced by `krb_mk_req`.

RD_AP_TIME
: The difference between the time at which the message, *in*, was produced by `krb_mk_req` and the time at which it was read by `krb_rd_req` is too large. The time difference must be within five minutes.

Restrictions

The caller of the functions, `krb_rd_req` and `krb_rd_safe`, must check the time order of messages and protect against replay attempts.

Files

```
/usr/include/krb.h
/usr/lib/libkrb.a
/usr/include/des.h
/usr/lib/libdes.a
/etc/srvtab
```

See Also

des_crypt(3krb), krb_sendmutual(3krb), krb_sendauth(3krb), krb_svc_init(3krb), krb_set_tkt_string(3krb), krb.conf(5krb)

krb_get_lrealm (3krb)

Name

krb_get_lrealm – Host/realm identification routines.

Syntax

```
#include <krb.h>
#include <des.h>

krb_get_lrealm (realm, n)
char *realm;
int  n;

char *krb_get_phost (alias)
char *alias;
```

Arguments

alias	Identifies a host whose name is to be converted to an instance name. The *alias* string is overwritten with the instance name. The *alias* string must be stored in a buffer of at least INST_SZ characters.
realm	Identifies a specific realm.
n	Specifies a specific position in a series of Kerberos hosts; must be set to 0.

Description

The routines of `krb_get_lrealm` allow an application to obtain information on host/realm relationships in a Kerberos network. The routines of this library are:

krb_get_phost
Converts the hostname pointed to by *alias*, which can be either an official name or an alias, into the instance name to be used in obtaining Kerberos tickets.

krb_get_lrealm
Initializes *realm* with the *n*th realm of the local host. The argument *realm* should be large enough to contain the maximum realm name determined by the constant REALM_SZ. The local realm name is stored in the `/etc/krb.conf` file. See the `krb.conf(5krb)` reference page.

Files

`/etc/krb.conf`

See Also

kerberos(3krb), krb.conf(5krb)

Name

krb_sendauth, krb_recvauth – Kerberos authentication library routines.

Syntax

```
#include <krb.h>
#include <des.h>
#include <netinet/in.h>

int krb_sendauth (options, fd, tkt_authen, f_service,
                  f_inst, f_realm, checksum, msg_data,
                  cred, schedule, l_addr, f_addr,
                  version_in)
long                options;
int                 fd;
KTEXT               tkt_authen;
char                *f_service;
char                *f_instance;
char                *f_realm;
u_long              checksum;
MSG_DAT             *msg_data;
CREDENTIALS         *cred;
Key_schedule        schedule;
struct sockaddr_in  *l_addr;
struct sockaddr_in  *f_addr;
char                *version_in;

int krb_recvauth (options, fd, tkt_authen_out, l_service,
                  l_instance, f_addr, l_addr, ad,
                  srvtab_file, schedule, version_out)
long                options;
int                 fd;
KTEXT               tkt_authen_out;
char                *l_service;
char                *l_instance;
struct sockaddr_in  *f_addr;
struct sockaddr_in  *l_addr;
AUTH_DAT            *ad;
char                *srvtab_file;
Key_schedule        schedule;
char                *version_out;
```

Arguments

options Defined in `/usr/include/krb.h`. To specify multiple options, construct the *options* argument as a bitwise-OR of the desired options. The options are as follows:

KOPT_DONT_MK_REQ
: `krb_sendauth` will not use the `krb_mk_req` function (see `kerberos(3krb)`) to produce the ticket-authenticator pair, *authen_tkt*. Instead, the ticket-authenticator pair is read from the argument, *tkt_authen*.

KOPT_DONT_CANON
krb_sendauth will not convert the instance name, *f_instance*, to canonical form. If KOPT_DONT_CANON is not set, the instance name used is the output from krb_get_phost (see krb_get_lrealm(3krb)) with argument *f_instance* as input.

KOPT_DO_MUTUAL
krb_sendauth and krb_recvauth provide authentication on both ends of the network connection. Otherwise, the caller of krb_sendauth is authenticated to the caller of krb_recvauth, but the caller of krb_recvauth is not authenticated to the caller of krb_sendauth. For mutual authentication to occur, both krb_sendauth and krb_recvauth must be called with this option set.

f_service Character pointer to the primary name of the foreign principal. The local principal is the principal that calls the above routines. The foreign principal is the principal with which the local principal is attempting to communicate. If KOPT_DONT_MK_REQ is set and KOPT_DO_MUTUAL is not, then *f_service* should be set equal to the NULL pointer.

f_instance
Character pointer to the instance name of the foreign principal. If KOPT_DONT_MK_REQ is set and KOPT_DO_MUTUAL is not, then *f_instance* should be set equal to the NULL pointer.

f_realm Character pointer to the realm name of the foreign principal. If the *f_realm* parameter is set equal to the NULL pointer, then the local realm is used as the *f_realm*. If KOPT_DONT_MK_REQ is set and KOPT_DO_MUTUAL is not, then *f_service* should be set equal to the NULL pointer.

l_service Character pointer to the primary name of the local principal.

l_instance
Character pointer to the instance name of the local principal.

fd The file descriptor used to send data to the foreign principal, or the file descriptor from which data from the foreign principal can be read. In either case, the file descriptor must be associated with a socket that uses blocking I/O.

tkt_authen
Pointer to the text structure in which the Kerberos library routines build the ticket-authenticator pair. This structure is designed to be included within the krb_sendauth message sent to the foreign principal to authenticate the local principal's identity to the foreign principal. This structure can be either input to krb_sendauth or output from krb_sendauth depending on whether KOPT_DONT_MK_REQ is set or not set. In either case, storage must be allocated for *tkt_authen*.

tkt_authen_out
Pointer to the ticket-authenticator pair that krb_recvauth reads from within the krb_sendauth message. The krb_sendauth message is sent by krb_sendauth to the local principal to authenticate the foreign principal to the local principal. Storage must be allocated for *tkt_authen_out*.

checksum

Input to `krb_sendauth`; *checksum* is packaged in the `krb_sendauth` message that is sent to the foreign principal. It serves as a secret piece of data that can only be known to the foreign principal if the foreign principal is authenticated as the foreign principal. It is used to facilitate mutual authentication, so if the KOPT_DO_MUTUAL is not set, the value of this argument is inconsequential. If both KOPT_DONT_MK_REQ and KOPT_DO_MUTUAL are set, then the *checksum* parameter must be equal to the checksum value used by `krb_mk_req` in the creation of the ticket-authenticator pair, *authen_tkt*.

msg_data

Pointer to a structure which is filled with the mutual authentication message sent by `krb_recvauth` and interpreted by `krb_sendauth`. The message sent from `krb_sendauth` to `krb_recvauth`, the message that includes the ticket-authenticator pair, authenticates only the caller of `krb_sendauth` to the caller of `krb_recvauth`. An additional message, the one returned by `krb_sendauth` inside *msg_data*, must be sent by `krb_recvauth` and interpreted by `krb_sendauth` in order to authenticate the caller of `krb_recvauth` to the caller of `krb_sendauth`. If the KOPT_DO_MUTUAL option is set, space must be allocated for the *msg_data* structure. Otherwise, since no message will be sent from `krb_recvauth` to `krb_sendauth`, the *msg_data* parameter should be set equivalent to the NULL pointer.

cred a pointer to a credentials structure that is output from `krb_sendauth`. The credentials structure includes the ticket that the local principal uses to authenticate to the foreign principal as well as other authentication information associated with the foreign principal. If the KOPT_DO_MUTUAL option is set, space must be allocated for the *cred* structure and the *cred* structure will be filled in by `krb_sendauth`. Otherwise, the *cred* structure will not be filled in by `krb_sendauth`, so the *cred* parameter should be set equivalent to the NULL pointer.

schedule a key schedule, derived from the session key between the local and foreign principals, that is output from `krb_sendauth` and `krb_recvauth`. If the KOPT_DO_MUTUAL option is set, the key schedule will be filled in; otherwise, the key schedule will not be filled. In any case, space must be allocated for the key schedule.

f_addr the address of the socket that the foreign principal is using to communicate with the local principal. If the KOPT_DO_MUTUAL option is not set on a call to `krb_sendauth`, then the *f_addr* parameter should be set equivalent to the NULL pointer. *f_addr* should never be set to NULL on a call to `krb_recvauth`.

l_addr the address of the socket that the local principal is using to communicate with the foreign principal. If the KOPT_DO_MUTUAL option is not set, the *l_addr* parameter should be set equivalent to the NULL pointer.

ad a pointer to the AUTH_DAT structure that describes the authentication association between the local and foreign principals. Since it is output from `krb_recvauth`, space for the *ad* structure must be allocated.

srvtab_file
: path name of the file that contains the key of the principal obtaining a ticket. If this value is set equal to a string of zero length, `srvtab_file[0]='\0'`, the default service table file (srvtab) value is used. If this value is set equal to the NULL pointer, then the key of the service is not read from the srvtab file, but is read from storage space internal to the libraries. The *srvtab_file* parameter cannot be set to the NULL string on the first call to `krb_sendauth`. The default srvtab file value is set to `/etc/srvtab` although this value can be changed by a call to the `krb_set_srvtab_string` function (see `krb_set_tkt_string(3krb)`).

version_in
: An application-specific version string input to `krb_sendauth`. This argument allows the caller of `krb_sendauth` to pass an application-specific version string, within the `krb_sendauth` message format, that the caller of `krb_recvauth` can use to match against its own version string. The version string can be up to KRB_SENDAUTH_VLEN characters long and, in addition, it can be set equal to the NULL string.

version_out
: An application-specific version string output from `krb_recvauth`. This argument allows the caller of `krb_recvauth` to receive the application-specific version string included in the `krb_sendauth` message that was sent by the foreign principal. The version string can be up to KRB_SENDAUTH_VLEN characters long.

Description

The `krb_sendauth(3krb)` routines are designed to be used by applications that communicate over a network, require the authentication of both parties accross the communications path, and which support "on-the-wire" protocols that have no room for authentication information. The `krb_sendauth(3krb)` routines are designed to perform only the authentication of the first message sent between such applications. Therefore, the `krb_sendauth(3krb)` routines should be used before any other communication occurs between the authenticating principals.

After the communications channel between the applications has been established, but before any communication takes place, and before the "on-the-wire" protocol of the application comes into effect, `krb_sendauth` creates a message which can authenticate the caller of `krb_sendauth`, "A", to the caller of `krb_recvauth`, "B". `krb_sendauth` then sends the message to "B" where it is read from the communications channel by `krb_recvauth`.

Next, `krb_recvauth` attempts to authenticate "A" by producing a response to "A" which, depending upon the value of KOPT_DO_MUTUAL and the success of the authentication of "A" by `krb_recvauth`, will contain either an error code, a code indicating success, or a mutual authentication message. `krb_recvauth` sends the response and returns to "B". `krb_sendauth` receives the message from "B", tries to authenticate "B" if KOPT_DO_MUTUAL is set, and then returns to "A".

Since the authentication information is sent between the applications before the "on-the-wire" protocol of the application comes into effect, the application must develop some method of distinguishing between the new authenticated initial message exchange and an old unauthenticated initial message exchange.

The krb_sendauth(3krb) routines make extensive use of the locally defined data types KTEXT, MSG_DAT, CREDENTIALS, and Key_schedule. For specific information on the definitions of these data types, see the des.h and krb.h files.

The routines found in the krb_sendauth(3krb) library are krb_sendauth and krb_recvauth:

krb_sendauth

The krb_sendauth function is designed to authenticate a local principal, "A", to the principal specified by the *f_service*, *f_instance*, and *f_realm* parameters, "B", and to allow the authentication of "B" to "A" as well. krb_sendauth uses file descriptor *fd*, to send the authentication message that will authenticate "A" to principal "B". It returns, in the *tkt_authen* parameter, the ticket-authenticator pair used to authenticate "A" to "B". The *version_in* parameter contains an application-specific version string which is transmitted to "B" along with the authentication message.

If mutual authentication is selected as an option, the file descriptor, *fd* will be used to receive a mutual authentication message from "B". To allow the mutual authentication to take place, *l_addr* and *f_addr* must be set equal to the address of the sockets which the local and foreign principals use to communicate. A value known only to "A" must be input to krb_sendauth as the *checksum* parameter. As the result of mutual authentication, *cred* will be filled with data describing the authentication information associated with "B", *schedule* will be set equal to the key_schedule of the session key between "A" and "B", and *msg_data* will be set equal to the mutual authentication message sent from "B" to "A".

fd must be a file descriptor associated with a blocking socket. Otherwise, krb_sendauth will not function correctly.

If "A" has been correctly authenticated to "B" and mutual authentication was not chosen as an option, or if "A" has been correctly authenticated to "B", and "B" correctly authenticated to "A" and mutual authentication was chosen as an option, then KSUCCESS is returned by krb_sendauth.

The following is a list of most of the error values from krb_sendauth. Since krb_sendauth calls other section 3 Kerberos routines (3krb) to perform its function, some of the error codes are references to the error codes of other functions:

SENDAUTH_OPNOTSUP
: The *options* bits sent to krb_sendauth contain a bit which is set, but does not correspond to an option.

SENDAUTH_WR
: krb_sendauth could not write the authentication message to "B" using *fd*.

KFAILURE
: The /etc/krb.conf file cannot be opened, or
The /etc/krb.conf file (see krb.conf(5krb)) is not formed properly, or
An authentication message was sent from "A" to "B", but "B" could not successfully identify "A", or
A mutual authentication message was sent from "B" to "A", but "A" could not successfully identify "B".

-1
: Negative one is returned if each byte of the session key does not have odd parity.

-2	Negative two is returned if the session key is a weak key as defined by `des_is_weak_key` (see `des_crypt(3krb)`).
NO_TKT_FIL	The ticket file does not exist.
TKT_FIL_ACC	The ticket file cannot be opened or the ticket file cannot be accessed.
TKT_FIL_LCK	The ticket file could not be locked for access.
TKT_FIL_FMT	The ticket file format is incorrect.
AD_NOTGT	There is no ticket-granting-ticket in the ticket file that can be used to ask for a ticket to communicate with the foreign principal.
SKDC_CANT	A Kerberos server must be contacted in order for `krb_sendauth` to perform its function, but the attempt cannot be made because a socket cannot be opened or bound, or there is no Kerberos server listed in `/etc/krb.conf`.
SKDC_RETRY	A Kerberos server needs to be contacted, but none responded even after several retries.
INTK_PROT	Kerberos protocol error.
GC_NOTKT	Information concerning the foreign principal does not exist in the ticket file.
RECVMUT_OPNOTSUP	The *options* bits sent to `krb_recvmutal` (see `krb_sendmutual(3krb)`) contain a bit which is set, but does not correspond to an option.
RECVMUT_RD	If the message cannot be read from the file descriptor *fd*, SENDMUT_RD is returned.
RD_AP_VERSION	If the Kerberos version used to create the mutual authentication message is not supported by `krb_recvmutual`, then RD_AP_VERSION is returned.
RD_AP_MSG_TYPE	If the message read from the file descriptor, *fd*, is not a mutual authentcation message, RD_AP_MSG_TYPE is returned.
RD_AP_MODIFIED	If the mutual authentication message has been modified between the "B" and "A" or it was in some way incorrectly produced, RD_AP_MODIFIED is returned.
RD_AP_TIME	Returned if the mutual authentication message is too old.

krb_recvauth

The `krb_recvauth` function is designed to wait for a message from `krb_sendauth` on the file descriptor *fd*, receive the message and attempt to authenticate the foreign principal, "A", to the local principal determined by the *l_service* and *l_instance* parameters. The *srvtab_file* must contain the private key of principal "B". The *tkt_authen_out* parameter is filled with the ticket-authenticator pair sent within the `krb_sendauth` message received by "B" from "A". *ad* is filled with information that describes the authentication association between "A" and "B". *version_out* is filled with the application version string included in the

krb_sendauth message.

If mutual authentication is selected as an option, the file descriptor *fd*, will be used to send a mutual authentication message to "A". To allow the mutual authentication to take place, *l_addr* and *f_addr* must be set equal to the address of the sockets that the local and foreign principals are using to communicate. As the result of mutual authentication, *schedule* will be set equal to the key_schedule of the session key between "A" and "B".

fd must be a file descriptor that is associated with a blocking socket. Otherwise, krb_recvauth will not function correctly.

If "A" has been correctly authenticated to "B" and mutual authentication was not chosen as an option, or if mutual authentication is an option and "A" has been correctly authenticated to "B" and "B" has sent a mutual authentication message to "B", then KSUCCESS is returned by krb_recvauth.

The following is a list of most of the error values from krb_recvauth. Since krb_recvauth calls other section 3 Kerberos routines (3krb) to perform its function, some of the error codes are references to the error codes of other functions.

RECVAUTH_OPNOTSUP
: The *options* bits sent to krb_recvauth contain a bit which is set but does not correspond to an option.

RECVAUTH_RD
: krb_recvauth could not read the authentication message sent to "B" using *fd*.

RECVAUTH_TKTLEN
: The length of the ticket-authenticator pair within the krb_sendauth message is longer than the maximum or less than or equal to 0.

RD_AP_VERSION
: The versions of Kerberos used by the caller of krb_sendauth is incompatible with the krb_recvauth version.

RD_AP_MSG_TYPE
: The ticket-authenticator pair given to krb_recvauth was not really a ticket-authenticator pair.

RD_AP_UNDEC
: The ticket could not be decyphered. This error can be caused by a forged or modified message.

RD_AP_INCON
: The message given to krb_recvauth contains an internal inconsistency. This could occur if the ticket in the ticket-authenticator pair does not match the authenticator.

RD_AP_BADD
: The ticket-authenticator pair cannot be used to authenticate a principal from the address specified by *f_addr*.

RD_AP_TIME
: The authenticator in the ticket-authenticator pair is too old to be used to authenticate the foreign principal.

RD_AP_NYV
: The time at which the ticket of the ticket-authenticator pair was created is too far ahead of the time of the local host of the local principal.

RD_AP_EXP
: The ticket is too old to be used.

-1	Negative one is returned if the each byte of the session key does not have odd parity.
-2	Negative two is returned if the session key is a weak key as defined by `des_is_weak_key`.
SENDMUT_OPNOTSUP	The options bits sent to `krb_sendmutal` contains a bit which is set but does not correspond to an option.
SENDMUT_MAKMSG	If there is an error in forming the mutual authentication message itself, SENDMUT_MAKMSG is returned.
SENDMUT_WR	If the mutual authentication message cannot be written to the file descriptor *fd*, SENDMUT_WR is returned.

Restrictions

`krb_sendauth` and `krb_recvauth` will not work properly on sockets set to nonblocking I/O mode.

See Also

kerberos(3krb), krb_sendmutual(3krb), krb_svc_init(3krb), des_crypt(3krb, krb_get_lrealm(3krb), krb_set_tkt_string(3krb), krb.conf(5krb).

Name

krb_sendmutual, krb_recvmutual – Kerberos mutual authentication routines

Syntax

```
#include <krb.h>
#include <des.h>

int krb_sendmutual (options, msg_out, success, fd,
                    f_addr, l_addr, ad, schedule)

long                options;
KTEXT               msg_out;
int                 success;
int                 fd;
struct sockaddr_in  *f_addr;
struct sockaddr_in  *l_addr;
AUTH_DAT            *ad;
Key_schedule        schedule;

int krb_recvmutual (options, fd, checksum, msg_in,
                    msg_data, cred, schedule, l_addr,
                    f_addr)

long                options;
int                 fd;
u_long              checksum;
KTEXT               msg_in;
MSG_DAT             *msg_data;
CREDENTIALS         *cred;
Key_schedule        schedule;
struct sockaddr_in  *l_addr;
struct sockaddr_in  *f_addr;
```

Arguments

options defined in `/usr/include/krb.h`. There is only one option currently supported, KOPT_NORDWR. If this option is not set, the mutual authentication information is read either from the supplied file descriptor, *fd*, or sent accross the supplied file descriptor, *fd*. If it is specified, then no data is read from or written to the file descriptor; instead, data is read from and written to the buffers supplied as parameters, *msg_in*, *msg_out*.

fd the file descriptor used to send data to the foreign principal, or it is the file descriptor from which data from the foreign principal can be read.

The foreign principal is the principal to which the principal that calls a `krb_sendmutual(3krb)` routine, the local principal, is attempting to mutually authenticate itself. The file descriptor must be associated with a socket that uses blocking I/O. The *fd* parameter is not used if the KOPT_NORDWR option is set.

f_addr the address of the socket that the foreign principal uses to communicate with the local principal.

l_addr the address of the socket that the local principal uses to communicate with the foreign principal.

msg_out If KOPT_NORDWR is sent as an option, *msg_out* is used as a buffer to store the mutual authentication data that should be sent to the foreign principal. If KOPT_NORDWR is not set, *msg_out* is not used and the mutual authentication message is written to *fd*.

success If success is not set to KSUCCESS, then the mutual authentication message generated by `krb_sendmutual` is a message indicating failure. This parameter is useful if the initial attempt to authenticate the foreign principal failed. Since this initial authentication attempt failed, then the attempt to authenticate the local principal to the foreign principal also must fail. If *success* is set to KSUCCESS, then a mutual authentication message is generated.

ad a pointer to the AUTH_DAT structure that describes the authentication association between the local and foreign principals. The *ad* structure is output from `krb_rd_req` (see `kerberos(3krb)`) and is used as input to `krb_sendmutual`. Space for the *ad* structure must be allocated.

checksum input to `krb_recvmutual`, it must have the same value as the *checksum* used as input to `krb_mk_req` (see `kerberos(3krb)`) or to `krb_sendauth` (see `krb_sendauth(3krb)`). The checksum is included in the ticket-authenticator pair produced by `krb_mk_req` and sent by `krb_sendauth` to the foreign principal. It serves as a secret piece of data that can only be known to the foreign principal if the foreign principal was authenticated as the foreign principal. It is included by `krb_sendmutual` in the mutual authentication message. If the checksum input to `krb_recvmutual` matches the one sent back by `krb_sendmutual`, then the caller of `krb_sendmutual` is authenticated to the caller of `krb_recvmutual`.

msg_in If KOPT_NORDWR is sent as an option, then data in *msg_in* is read as if it contained the mutual authentication bits sent to the local principal by the foreign principal. If KOPT_NORDWR is not set, then *msg_in* is not used and the mutual authentication message is read from *fd*.

msg_data a structure returned by `krb_recvmutal` that is filled with the mutual authentication message sent to the local principal as well as information about the status of the message. Space must be allocated for the *msg_data* structure.

cred a pointer to a credentials structure that is input to `krb_recvmutual`. The credentials structure that *cred* points to must be the credentials structure that includes the ticket that the local principal uses to authenticate the foreign principal. This credential structure is usually obtained through the use of `krb_get_cred` (See `kerberos(3krb)`).

schedule the key schedule derived from the session key between the local and foreign principals. It is input to both `krb_sendmutual` and `krb_recvmutual`, and it can be generated from the session key with `des_key_sched` (see `des_crypt(3krb)`).

Description

The krb_sendmutual(3krb) routines are designed to be used by applications which communicate over the network, support "on-the-wire" protocols in which authentication information can be placed, and require both parties in the communications process to be authenticated to the other (mutual authentication). They are best used with krb_mk_req and krb_rd_req. If a principal "A" calls krb_mk_req and sends the output to principal "B", which uses krb_rd_req to interpret the data successfully, then "B" will have authenticated principal "A". But, principal "A" will not know that the message it sent was really received by "B". To prove the identity of principal "B" to principal "A" after the calls to krb_mk_req and krb_rd_req are finished, the krb_sendmutual(3krb) calls are used.

krb_sendmutual and krb_recvmutual can also be used with krb_mk_req and krb_rd_req by applications which cannot tolerate additions to their "on-the-wire" protocols. After the communications channel between "A" and "B" is established, but before "A" and "B" communicate and before the "on-the-wire" protocol of the applications comes into effect, krb_mk_req and krb_rd_req can be used as described above to authenticate "A" to "B". krb_sendmutual and krb_recvmutual can then be used with the KOPT_NORDWR option not set to authenticate "B" to "A".

Since the authentication information is sent between the applications before the "on-the-wire" protocol of the application comes into effect, the application must develop some way to distinguish between the new authenticated initial message exchange and an old unauthenticated initial message exchange. This is not a recommended use for krb_sendmutual and krb_recvmutual. If you do not want to modify the "on-the-wire" protocol of an application, yet want to authenticate the application, then use the krb_sendauth(3krb) routines.

The routines of this library make extensive use of the following locally defined data types: KTEXT, AUTH_DAT, CREDENTIALS, Key_schedule, and MSG_DAT. For more specific information on the definitions of these data types, see the des.h and krb.h files.

krb_sendmutual

krb_sendmutual is used to produce and possibly send the data that will authenticate principal "B" to principal "A". If the authentication of principal "A" did not succeed, *success* should be set to KFAILURE, and krb_sendmutual produces a message indicating authentication failure. If it is set to KSUCCESS, then krb_sendmutual produces the data necessary to authenticate "B" to "A". If the option KOPT_NORDWR is set, the data is written to buffer *msg_out*; otherwise, it is written to file descriptor, *fd*.

The following is a list of the return values and, if they are error codes, their possible cause:

SENDMUT_OPNOTSUP
: The *options* bits sent to krb_sendmutal contain a bit that is set but does not correspond to an option.

SENDMUT_PARAM
: The *msg_out* structure must have space within it allocated to store the message. Otherwise, SENDMUT_PARAM is returned if the KOPT_NORDWR option is set.

SENDMUT_MAKMSG
If there is an error in forming the mutual authentication message itself, SENDMUT_MAKMSG is returned.

SENDMUT_WR If the message cannot be written to the file descriptor *fd*, SENDMUT_WR is returned.

KSUCCESS If the message has been correctly formed, KSUCCESS is returned.

krb_recvmutual

The `krb_recvmutual` routine interprets the mutual authentication message sent to principal "A" by principal "B". If the KOPT_NORDWR option is set, `krb_recvmutual` reads from buffer *msg_in*, the message sent from "B" to "A". Otherwise, it reads the message from file descriptor, *fd*. The *checksum* sent as input to `krb_recvmutual` must be the same checksum used as input to `krb_mk_req`. The checksum is an integral part of proving the identity of principal "B" to "A". The following is a list of the return values and, if they are error codes, their possible cause:

RECVMUT_OPNOTSUP
The *options* bits sent to `krb_recvmutal` contain a bit that is set, but does not correspond to an option.

RECVMUT_MSGLEN
The size of the *msg_in* buffer is incorrect.

RECVMUT_RD If the message cannot be read from the file descriptor *fd*, then SENDMUT_RD is returned.

RD_AP_VERSION If the Kerberos version used to create the mutual authentication message is not currently supported by `krb_recvmutual`, then RD_AP_VERSION is returned.

RD_AP_MSG_TYPE
If the message that is read from the file descriptor *fd*, or input as *msg_in* is not a mutual authentication message, RD_AP_MSG_TYPE is returned.

RD_AP_MODIFIED
If the message has been modified between principals "B" and "A", or if was incorrectly produced, then RD_AP_MODIFIED is returned.

RD_AP_TIME If the mutual authentication message is too old, RD_AP_TIME is returned.

KFAILURE If principal "A" was not authenticated to principal "B", or if the mutual authentication message fails to identify "B", KFAILURE is returned.

KSUCCESS If principal "B" has been correctly authenticated to principal "A", KSUCCESS is returned.

Restrictions

`krb_sendmutal` and `krb_recvmutal` will not work properly with sockets that do not use blocking I/O.

See Also

kerberos(3krb), krb_sendauth(3krb), des_crypt(3krb), krb_svc_init(3krb)

krb_set_tkt_string (3krb)

Name

krb_set_tkt_string, krb_set_srvtab_string – Environmental setup of the Kerberos libraries

Syntax

```
#include <krb.h>

void krb_set_tkt_string (filename)
char *filename

void krb_set_srvtab_string (filename)
char *filename
```

Arguments

filename The filename of the Kerberos ticket cache file or the name of the service table file.

Description

The `krb_set_tkt_string` routine sets the default name of the file that holds a cache of service tickets and associated session keys belonging to a Kerberos principal. The routine accepts a filename for the cache and copies this name into the local storage of `libkrb`. The default before any calls to `krb_set_tkt_string`, is `/var/dss/kerberos/tkt/tkt[`*uid*`]` where `uid` is the user ID of the process that calls `krb_set_tkt_string`.

You should call `krb_set_tkt_string` during Kerberos initialization to assure that any routines called later receive the proper name if they require the filename of the cache.

The `krb_set_srvtab_string` routine sets the default name of the file that stores the keys of the Kerberos applications running on the local host. The routine accepts a filename for the service table file and copies this name into the local storage of `libkrb`.

You should call `krb_set_srvtab_string` during the Kerberos initialization of a service to assure that any subsequently called routines that require the filename of the service table receive the proper name. The default, before any calls to the `krb_set_srvtab` string, is `/etc/srvtab`.

Files

`/var/dss/kerberos/tkt/tkt[`*uid*`]`

`/etc/srvtab`

See Also

kerberos(3krb), krb_sendauth(3krb), krb_sendmutual(3krb)

Name

krb_svc_init, krb_get_svc_in_tkt, krb_get_pw_in_tkt – Kerberos authentication initialization routines

Syntax

```
#include <krb.h>
#include <des.h>

krb_svc_init (user, instance, realm, lifetime,
              srvtab_file, tkt_file)
char  *user, *instance, *realm;
int   lifetime;
char  *srvtab_file, *tkt_file;

krb_get_svc_in_tkt (user, instance, realm, service,
                    service_instance, lifetime,
                    srvtab_file)
char  *user, *instance, *realm, *service,;
char   *service_instance;
int   lifetime;
char  *srvtab_file;

krb_get_pw_in_tkt (user, instance, realm, service,
                   service_instance, lifetime,
                   password)
char  *user, *instance, *realm,;
char   *service, *service_instance;
int   lifetime;
char  *password;
```

Arguments

user For `krb_get_svc_in_tkt` and `krb_get_pw_in_tkt`, the primary name of the principal that is obtaining a ticket that will authenticate it to principal, *service*. For `krb_svc_init`, the primary name of the principal that is obtaining a ticket to communicate with the ticket-granting service.

instance For `krb_get_svc_in_tkt` and `krb_get_pw_in_tkt`, the instance name of the principal that is obtaining a ticket that will authenticate it to principal, *service*. For `krb_svc_init`, the instance name of the principal that is obtaining a ticket to communicate with the ticket-granting service.

realm For `krb_get_svc_in_tkt` and `krb_get_pw_in_tkt`, the realm name of the principal that is obtaining a ticket that will authenticate it to principal, *service*. For `krb_svc_init`, the realm name of the principal that is obtaining a ticket to communicate with the ticket-granting service.

service The primary name of the service for which a ticket will be obtained.

service_instance
The instance of the service for which a ticket will be obtained.

lifetime The number of five-minute intervals for which the obtained ticket should

be valid. Values greater than 255 will be set to 255. Values greater than the maximum lifetime allowed for tickets given to the requesting principal will be set to the maximum lifetime allowed. The maximum lifetime of the tickets granted to a principal is determined when the principal is added to the Kerberos database.

srvtab_file The path name of the file that contains the key of the principal obtaining a ticket. If this value is set to the NULL pointer, the default service table (srvtab) file value is used. The default srvtab file value is set by default to `/etc/srvtab`, although this value can be changed by a call to the `krb_set_srvtab_string` function. (Refer to `krb_set_tkt_string(3krb)`).

tkt_file The path name of the file into which the credentials and tickets of the user or service should be placed. If the *tkt_file* parameter is equal to the NULL pointer, then the default ticket file value is used. The default ticket file value is set equal to `/var/dss/kerberos/tkt/tkt.`[*uid*] where `uid` is the user ID of the process that calls the above functions. The default ticket file value can be changed by the `krb_set_tkt_string(3krb)` function call.

password The password of the principal that is obtaining a ticket that will authenticate it to principal, *service*. If the password input is the NULL string, then `krb_get_pw_in_tkt` will prompt for a password on `stdout` and read the password from `stdin`.

Description

The `krb_svc_init(3krb)` routines are designed to obtain for the requesting principal a ticket to communicate with a specific service. They require that the password/key of the requesting principal be either available as an argument, or available from the *srvtab_file* argument or from `stdin`. Since the `krb_svc_init(3krb)` routines always require a password, they are best used to obtain the ticket used to communicate with the ticket-granting service. The ticket-granting ticket is used by the other Kerberos routines to obtain tickets to communicate with principals other than the ticket-granting service, without needing the key of the principal.

The `krb_sendauth(3krb)` routines as well as the `kerberos(3krb)` routines will not work as intended without the presence of a ticket-granting ticket.

The routines of `krb_svc_init(3krb)` are as follows:

krb_svc_init

For the principal with a primary name of *user*, an instance name of *instance*, and a realm name of *realm*, the `krb_svc_init` routine obtains a ticket that the principal can use to communicate with the ticket-granting service. The key of the principal is read from *srvtab_file* and the ticket obtained is placed in *tkt_file*.

If the *realm* argument is equivalent to the NULL string, then the realm of which the local host is a member, is used by default. If *lifetime* is equivalent to 0, then the default lifetime, 255, is used. If *srvtab_file* is not equivalent to the NULL string, then the *srvtab_file* parameter is used as the service table (srvtab) file name and the default srvtab file is set equal to the *srvtab_file* parameter. If *srvtab_file* is equivalent

to NULL, then the default srvtab file is used. If the *tkt_file* parameter is not equivalent to the NULL string, then the *tkt_file* parameter is used as the ticket file name and the default ticket file is set equal to the *tkt_file* parameter. If the *tkt_file* parameter is NULL, then the default ticket file value is used.

`krb_svc_init` returns INT_OK if `krb_svc_init` has successfully obtained a ticket-granting ticket. The following is a list of most of the error values returned from `krb_svc_init` and their possible cause:

KFAILURE
: The `/etc/krb.conf` file (see `krb.conf(5krb)`) cannot be opened or it is not properly formed, or
The service table (srvtab) file does not exist, or
A read of the srvtab file failed, or
The srvtab file is badly formatted, or
The srvtab file did not contain the key of the principal with primary name, *user*, or
A write to the ticket file failed.

SKDC_CANT
: A Kerberos server must be contacted so that `krb_svc_init` can perform its function, but the attempt cannot be made because a socket cannot be opened or bound, or there is no Kerberos server listed in `/etc/krb.conf`.

SKDC_RETRY
: A Kerberos server needs to be contacted, but none responded even after several attempts.

INTK_PROT
: Kerberos protocol version mismatch. The version of the Kerberos protocol supported by `krb_svc_init` does not match the Kerberos protocol version supported by the `kerberos(8krb)` daemon.

INTK_BADPW
: The ticket returned by the `kerberos` daemon did not decrypt correctly. This is usually caused by an incorrect service password.

INTK_ERR
: The ticket sent from the `kerberos` daemon was not a ticket to communicate with the ticket-granting service, or
The ticket file cannot be accessed, or
The ticket file could not be created, or
A write operation to the ticket file failed.

TKT_FIL_LCK
: The ticket file could not be locked for access.

krb_get_svc_in_tkt

For the principal with a primary name of *user*, an instance name of *instance* and a realm name of *realm*, the `krb_get_svc_in_tkt` routine obtains a ticket to communicate with the principal that has a primary name of *service* and an instance name of *service_instance*. The key of the requesting primary is read from the file *srvtab_file* and the tickets are placed in the default ticket file. If the *srvtab_file*

argument is equivalent to the NULL string, then the default srvtab file value is used instead of the *srvtab_file* parameter. The default srvtab file value and default ticket file value can be changed respectively by `krb_set_srvtab_sting` and `krb_set_tkt_string`. To obtain the ticket-granting ticket, the *service* parameter must be set equal to "krbtgt" and the *service_instance* argument must be set equal to the realm name of the local realm.

`krb_get_svc_in_tkt` returns INT_OK if `krb_get_svc_in_tkt` has successfully obtained a ticket to communicate with principal, *service*. The following is a list of most of the error values returned from `krb_get_svc_in_tkt` and their possible causes:

KFAILURE
: The `/etc/krb.conf` file cannot be opened or it is not properly formed, or
A read of the service table (srvtab) file failed, or
The srvtab file did not contain the key of the principal with primary name, *user*, or
A write to the ticket file failed.

SKDC_CANT
: A Kerberos server must be contacted in order for `krb_svc_init` to perform its function, but the attempt cannot be made because a socket cannot be opened or bound, or there is no Kerberos server listed in `/etc/krb.conf`.

SKDC_RETRY
: A Kerberos server needs to be contacted but none responded even after several attempts.

INTK_PROT
: Kerberos protocol version mismatch. The version of the Kerberos protocol supported by `krb_get_svc_in_tkt` does not match the Kerberos protocol version supported by the `kerberos` daemon.

INTK_BADPW
: The ticket returned by the `kerberos` daemon did not decrypt correctly. This is usually caused by an incorrect service password.

INTK_ERR
: The ticket sent from the `kerberos` daemon was not a ticket to communicate with the ticket-granting service, or
The ticket file cannot be accessed, or
The ticket file could not be created, or
A write operation to the ticket file failed.

TKT_FIL_LCK
: The ticket file could not be locked for access.

krb_get_pw_in_tkt

For the principal with a primary name of *user*, an instance name of *instance*, and a realm name of *realm*, the `krb_get_pw_in_tkt` routine obtains a ticket to communicate with the principal with a primary name of *service* and an instance name of *service_instance*. The key of the principal must be input either as the *password*

parameter or, if the password field is equivalent to the NULL string, the password must be input from `stdin`.

The tickets that are obtained are placed in the default ticket file. The default ticket file can be changed by the `krb_set_tkt_string` function. To obtain the ticket-granting ticket, the *service* parameter must be set equal to "krbtgt" and the *service_instance* argument must be set equal to the realm name of the local realm.

`krb_get_pw_in_tkt` returns INT_OK if `krb_get_pw_in_tkt` has successfully obtained a ticket to communicate with principal, *service*. The following is a list of most of the error values returned from `krb_get_pw_in_tkt` and their possible causes:

KFAILURE
: `/etc/krb.conf` file cannot be opened or it is not properly formed. A write to the ticket file failed.

SKDC_CANT
: A Kerberos server must be contacted in order for `krb_svc_init` to perform its function but the attempt cannot be made because a socket cannot be opened or bound, or there is no Kerberos server listed in `/etc/krb.conf`.

SKDC_RETRY
: A Kerberos server needs to be contacted but none responded even after several attempts.

INTK_PROT
: Kerberos protocol version mismatch. The version of the Kerberos protocol supported by `krb_get_pw_in_tkt` does not match the Kerberos protocol version supported by the `kerberos` daemon.

INTK_BADPW
: The ticket returned by the `kerberos` daemon did not decrypt correctly. This is usually caused by an incorrect user password.

INTK_ERR
: The ticket sent from the `kerberos` daemon was not a ticket to communicate with the ticket-granting service, or
The ticket file cannot be accessed, or
The ticket file could not be created, or
A write operation to the ticket file failed.

TKT_FIL_LCK
: The ticket file could not be locked for access.

See Also

krb_get_lrealm(3krb), krb_set_tkt_string(3krb), kerberos(3krb), krb_sendauth(3krb), kerberos(8krb)